##### **Functional Interface**

Functional interface is a kind of interface that has ****only 1 abstract method****.

Functional interface can have multiple default methods and static methods.

@FunctionalInterface // annotation is optionalpublic interface Doable {

// this is the only abstract method. It doesn't have body.

int doIt(String event);

default int defaultMethodDemo1(){ // this is a default method, it has body.

System.out.println("This is Java");

}

static int staticMethodDemo2() {. // this is a static method, it has body.

System.out.println("Static");

}

}

 Commonly used functional interfaces, they are all in JDK:

public interface Runnable {

public abstract void run();}

public interface Callable {

V call() throws Exception;}

// this interface has lots of static and default methods, but just has 1 abstract method.public interface Comparator {

int compare(T o1, T o2);}

@FunctionalInterfacepublic interface Function<T, R> {

R apply(T t);}

@FunctionalInterfacepublic interface Predicate<T> {

boolean test(T t);}

@FunctionalInterfacepublic interface Supplier<T> {

T get();}

@FunctionalInterfacepublic interface Consumer<T> {

void accept(T t);}

##### **Lambda**

Before Java 8, if we create an object from any interfaces, we need to implement a class and new the object.

For example, if we want to create a supplier in above code:

public class MySupplier implements Supplier<String> {

@Override

public String get() {

return "hello";

}}

MySupplier ms = new MySupplier();

or we can use Anonymous inner class:

Supplier ms1 = new Supplier(){

@Override

public String get() {

return "hello";

}};

The code is bulky and having lots of redundancy.

Lambda is introduced from Java 8 to implify the creation of objects from funcitonal interfaces.

Since all the functional interfaces have only 1 abstract method, the lambda acually is used to represent the implementation of that only 1 method.

Lambda doesn't have name, the ****parameter list**** and ****return type**** define the lambda.

Lambda looks like this:

() -> {}

() part is the parameter list of the abstract method.

{} is the body of the abstract method.

For example the Supplier interface has 1 abstract method, it takes no parameter and returns an Object. So we can use:

() -> {return "hello";}

If there is only 1 statement in the body.  We can simplify the return, just omit the {} and return keyword, like this:

() -> "hello"

Another example:

// Assume we have a customized Functional Interface

public interface Doable {

boolean doIt(String type, int number);

}

// the Above functional interface takes two parameter and return 1 boolean.// we can use lambda like

(String a, int b) -> {return true};// simplify the return and parameter list.(a,b) -> true;

// if multiple statements, we cannot omit the {}(a,b) -> { System.out.println("hi"); return true;}

Some commonly used Lambda:

// Comparator(a,b)-> a - b; // this can be used to compare two int.(a,b) -> a.getAge() - b.getAge(); // this can be used to compare two Person's age

// RunnableRunnable r = () -> { // code to be executed in new thread};

##### **Method Reference**

Method reference is used to use existing method to implement a functional interface. It is a short-cut of Lambda.

Consumer c = (i) -> System.out.println(i);// using method referenceConsumer c = System.out::prinln; // notice there is no ().